嵌入式 Forth 操作系统实时调度算法研究 *

黄忠建，代红兵†，王 蕾

(云南大学 信息学院 云南省高校数字媒体技术重点实验室，昆明 650223)

摘 要：针对目前嵌入式 Forth 操作系统中缺乏实时调度机制的问题，对基于 Forth 虚拟机架构的嵌入式操作系统中多任务调度的关键技术进行了研究。采用 Forth 虚拟机技术，新定义了一种中断任务类型来处理实时突发事件，并给出了一种新的任务调度算法来调度 Forth 系统中终端任务、后台任务以及中断任务顺利运行。实验结果表明，改进后的 Forth 系统能够通过实时调度处理突发事件，并且实时响应度高，尤其适合应用于对实时性有要求的嵌入式环境中，以满足日趋复杂的嵌入式环境对高效操作系统和 Forth 技术的应用需求。

关键词：Forth 系统；多任务；实时调度

中图分类号：TP316 doi: 10.3969/j.issn.1001-3695.2018.03.0237

Research on real-time scheduling algorithm for embedded Forth operating system

Huang Zhongjian, Dai Hongbing†, Wang Lei

(*Digital Media Technology Key Laboratory of Universities in Yunnan, School of Information Science & Engineering Yunnan University*, *Kunming* 650223, *China*)

Abstract: For the problem of lacking real-time scheduling mechanism in the embedded Forth operating system at present, this paper investigated the key technologies of multitask-scheduling of embedded operating system based on Forth virtual machine architecture. On the basis of Forth virtual machine technology, this paper defined a new interrupt task type to deal with real-time emergencies and proposed a new scheduling algorithm, which enabled the scheduling of terminal tasks, background tasks and interrupt tasks in Forth system to be successfully executed. The experimental results demonstrate that the improved Forth system can handle emergencies through real-time scheduling and has a high degree of real-time response and is especially suitable for the embedded environment which is required for real-time performance, so as to meet the increasingly complex application requirements of embedded system for efficient operating system and Forth technology.

Key words: Forth system; multitask; real-time scheduling

0 引言

1969 年美国的 Chuck Moore 发明了 Forth[1]。Forth 是一种操作系统、一套开发工具、一种高级程序设计语言、一种汇编语言以及一种软件设计准则的集合体[2]。近年来，国内外对 Forth 的研究主要集中于 Forth 系统构建和 Forth 处理器。自 Forth 诞生以来，经过不断的扩展和补充，已经有了多个 Forth 版本，比如 AmForth、SwiftX、PunyForth 等，其中 SwiftX[3]是目前最具代表性的版本。就 Forth 处理器而言，Edvin 等人[4]介绍了一种基于堆栈的 32 位 Forth 微处理器的设计，详细阐述了这种微处理器的架构以及指令设计等方面的内容。Hanna 等人[5]提出了一种称为 rForth 的 32 位 Forth 内核体系架构，其包括一个浮点运算单元、一个中断控制器和一个可访问的扩展内存。Ting 等人[6~11]对在 FPGA 上用 VHDL 设计 Forth 处理器进行了描述。此外，Chuck Moore 创办的 GreenArrays 公司目前已在其生产的 GA144 芯片上集成了 144 个 Forth 处理器[12]。

正如多任务调度是其他操作系统的核心功能一样，Forth 系统的核心功能也包括多任务调度。最早的 Forth 操作系统调度采用的是与传统操作系统相同的方式，即基于 CPU 方式的调度，这种方式任务切换需要保存和恢复 CPU 现场和 Forth 状态，例如早期的 Forth11 和 Forth88[13]。基于 CPU 方式的多任务调度存在的问题是开销太大，针对这个问题，基于虚拟机的 Forth 多任务操作系统（下文简称 Forth 系统）开始出现，其特点是简洁高效、对硬件层抽象、可重构、可扩展、可移植以及可交互，多任务基于堆栈调度，并且采用协作式轮循的调度方式，典型的有 ColorForth、AmForth、PunnyForth、SwiftX 等。这种基于虚拟机的 Forth 操作系统由于具备了诸多的优势开始成为研究热点，并逐渐成为主流。基于虚拟机的 Forth 多任务调度

收稿日期：2018-03-21；**修回日期：**2018-05-04 **基金项目：**国家自然科学基金资助项目（61640205）

作者简介：黄忠建（1993-），男，云南文山人，硕士研究生，主要研究方向为嵌入式操作系统；代红兵（1963-），男（通信作者），正高级工程师，硕士，主要研究方向为嵌入式系统、数字电视技术（hbdai_it@126.com）；王蕾（1994-），女，硕士研究生，主要研究方向为嵌入式操作系统．

成功解决了在任务切换时开销大的问题，但同时也带来了另一个问题：多任务调度是非实时的。

Forth 系统主要应用于嵌入式领域，对于一个应用于嵌入式领域的操作系统而言，能够及时处理各种突发事件的意义不言而喻。纵览目前国内外对 Forth 的研究，在基于虚拟机的实时调度方面依然是空白。因此，通过对目前基于虚拟机的 Forth 系统中多任务调度机制相关关键问题进行研究，本文给出了一种提升 Forth 系统多任务调度实时性的实现方法。

1　现有 Forth 系统协作式多任务调度机制

目前主流的 Forth 系统普遍支持协作式多任务机制，任务调度方式为轮询调度。Forth 系统提供终端任务和后台任务两种不同类型的任务，并且在数量上只有一个终端任务和允许用户建立多个后台任务。不管终端任务还是后台任务，均只有就绪和休眠两种状态，只有在就绪状态下的任务才能在任务调度时获得 CPU 的控制权。在单任务模式下，终端任务得到 CPU 的控制权。在多任务模式下，终端任务与后台任务通过协作式轮询的方式交替得到 CPU 的控制权。终端任务与后台任务是通过循环链表的方式组织起来的，如图 1 所示。



图 1　Forth 系统中终端任务与后台任务的组织方式

Sys-task 代表终端任务，也即系统任务，用于与用户交互。$Bg\text{-}task_1$~$Bg\text{-}task_{n-1}$ 代表后台任务，用于在后台处理其他事情。

在 Forth 系统多任务调度中，pause 是任务调度原语，它的功能在于：首先保存当前正在运行的任务的执行断点，然后去任务链里寻找下一个就绪的任务，最后把 CPU 控制权转交给已找到的就绪的任务，之后就绪的任务开始运行。如图 1 所示的任务链，假设终端任务正在运行，Forth 系统中协作式多任务的轮询调度的原理为：

a)终端任务正在运行，在运行的过程中遇到了任务调度原语 pause，终端任务去执行 pause。在 pause 中，终端任务首先保存其执行断点，然后去任务链里寻找下一个就绪的后台任务 $Bg\text{-}task_i$（$1 \le i \le n-1$），最后把 CPU 的控制权交给 $Bg\text{-}task_i$，随后 $Bg\text{-}task_i$ 开始运行。

b)$Bg\text{-}task_i$ 正在运行，在运行的过程中遇到了任务调度原语 pause，$Bg\text{-}task_i$ 去执行 pause。在 pause 中，$Bg\text{-}task_i$ 首先保存其执行断点，然后去任务链里寻找下一个就绪的后台任务 $Bg\text{-}task_j$（$i < j \le n-1$），最后把 CPU 的控制权交给 $Bg\text{-}task_j$，随后 $Bg\text{-}task_j$ 开始运行。

c)后台任务 $Bg\text{-}task_j$ 正在运行，在运行的过程中遇到了任务调度原语 pause，倘若在 $Bg\text{-}task_j$ 与后台任务 $Bg\text{-}task_{n-1}$ 之间已经没有就绪的任务，由于任务链是通过循环链表的方式组织起来的，那么 $Bg\text{-}task_j$ 在执行 pause 的过程中寻找下一个就绪任务的时候，会把 CPU 的控制权交给终端任务，终端任务继续从上次发生调度的断点开始往下执行。

d)终端任务在运行的过程中，通过执行 pause 的方式将 CPU 的控制权交给 $Bg\text{-}task_i$，$Bg\text{-}task_i$ 从上一次发生调度的断点继续往下执行；$Bg\text{-}task_i$ 在运行的过程中，通过执行 pause 的方式将 CPU 的控制权交给 $Bg\text{-}task_j$，$Bg\text{-}task_j$ 从上一次发生调度的断点继续往下执行。如此不断循环，直到 $Bg\text{-}task_i$ 与 $Bg\text{-}task_j$ 的执行体均执行完，之后系统进入单任务模式，终端任务获得 CPU 的控制权。

在 Forth 系统中，任务是竞争系统资源的最小运行单元。目前 Forth 系统只有终端任务和后台任务这两种不同类型的任务，在终端任务和后台任务（$Bg\text{-}task_1$~$Bg\text{-}task_{n-1}$）的执行期间，倘若外部有一个实时突发事件请求处理，现有 Forth 系统并没有实时调度机制去实时处理突发事件，这显然严重影响了 Forth 系统的实时性，同时也局限了 Forth 在对实时性有要求的嵌入式环境中的应用范围。

2　Forth 系统实时任务调度

2.1　Forth 系统实时任务调度机制

1)Forth 系统处理实时突发事件

在 Forth 系统协作式多任务轮询调度的基础上，结合中断技术，本文定义了**一种新的任务类型---中断任务类型**来处理外部实时突发事件。终端任务、后台任务以及中断任务是通过循环链表的方式组织起来的，加入了中断任务的新任务链表如下图 2 所示：

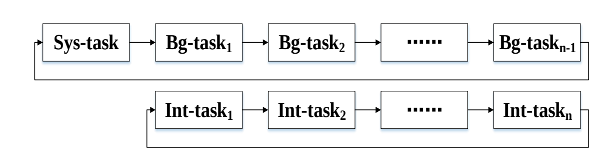


图 2　终端任务、后台任务以及中断任务的任务链组织方式

$Int\text{-}task_1$~$Int\text{-}task_n$ 代表中断任务。终端任务、后台任务以及中断任务的任务状态只有两种：就绪和休眠。中断任务处理突发事件的过程为：a)外部实时突发事件产生中断请求，当前正在执行的任务转去执行中断服务程序，在中断服务程序里就绪对应的中断任务；b)调度中断任务执行。

2)中断任务的调度

Forth 系统原有的任务调度原语 pause 并不能调度中断任务。按照图 2 所示的任务链组织方式，在原始任务调度原语 pause 的基础上，本文定义了**一个新的任务调度原语 INT-PAUSE** 来调度终端任务、后台任务以及中断任务，INT-PAUSE 调度的基本思想是：

a)引入临界资源 intTask 和 readyTask[]。intTask 初值为 0，用于记录当前中断任务链里已经就绪的中断任务的数量。readyTask[0]存储终端任务的 TCB 或者后台任务的 TCB，目的是当中断任务执行完的时候能够返回发生中断时的终端任务或者后台任务，使终端-后台任务的任务链里下一个就绪的任务运行。每当有中断请求就绪了中断任务，intTask 就加 1，且 readyTask[1]~ readyTask[intTask]里存储着在中断服务程序里就

绪的中断任务的 TCB 地址。

b)若终端任务或者后台任务正在运行的时候来了一个中断：(a)把当前正在运行的终端任务或者后台任务的 TCB 地址存储在 readyTask[0]里；(b)intTask 加 1,且 readyTask[intTask]里存储就绪的中断任务的 TCB 地址。若中断任务正在运行的时候来个一个中断：intTask 加 1,且 readyTask[intTask]里存储就绪的中断任务的 TCB 地址。

c)当前正在运行的任务每次执行 INT-PAUSE 进行任务调度的时候,若 intTask 大于 0，调度 readyTask[intTask]里存储的就绪的中断任务的 TCB 运行，然后 intTask 减 1；若 intTask 等于 0 并且 readyTask[0]等于 0，调度在终端-后台任务的任务链里下一个已经就绪的任务运行；若 intTask 等于 0 并且 readyTask[0]不等于 0，从 readyTask[0]里存储的终端任务的 TCB 或者后台任务的 TCB 开始，调度在终端-后台任务的任务链里下一个就绪的任务运行，然后将 readyTask[0]赋值为 0。

按照图 2 所示的任务链组织方式，某个就绪的中断任务 Int-task_k 从就绪到运行需要等待的时间为

$$T_{k-wait}=T_i+\sum_{m=k+1}^{m=intTask}T_{m\text{-}exec}+(intTask-k+1)*T_{INT-PAUSE}$$

其中:1≤k≤n，0≤i≤n-1，当 i=0 时，T_i 表示终端任务从发生中断的地方执行到任务调度原语 INT-PAUSE 的时间，当 i≠0 时，T_i 表示后台任务从发生中断的地方执行到任务调度原语 INT-PAUSE 的时间。$T_{m\text{-}exec}$ 为中断任务 Int-task_m 一次执行完其执行体的时间，且 k+1≤intTask。$T_{INT\text{-}PAUSE}$ 为任务调度原语 INT-PAUSE 的执行时间。

当 k+1>intTask 时，即 Int-task_k 表示中断任务链里最后一个就绪的任务，$T_{k-wait}=T_i+T_{INT-PAUSE}$。

当中断任务链里只有一个任务 Int-task_1 就绪时，$T_{1-wait}=T_i+T_{INT-PAUSE}$。

当中断任务里 n 个中断任务全部就绪时，$T_{n-wait}\le T_{k-wait}\le T_{1-wait}\ (1\le k\le n)$，即

$$T_i+T_{INT-PAUSE}\le T_{k-wait}\le T_i+\sum_{m=2}^{m=n}T_{m-exec}+n*T_{INT-PAUSE}。$$

由于在终端任务或者后台任务的执行体中何处才开始调度是由程序员人为加入 INT-PAUSE 决定的，因此 T_i 是不确定的，同时 T_m 也取决于中断任务执行体需要执行时间的长短。

2.2 Forth 系统实时调度算法

1)中断任务类型

(1)中断任务的任务控制块

加上本文创建的中断任务类型，Forth 系统目前能够提供三种类型的任务：终端任务、后台任务以及中断任务。其中，中断任务用于处理外部突发的事件。每个中断任务都拥有自己的任务控制块（TCB）。终端任务的 TCB、所有后台任务的 TCB 和所有中断任务的 TCB 都位于用户区里，这些 TCB 是按照图 2 所示的循环链表方式组织起来，通过本文新定义的任务调度原语 INT-PAUSE 进行调度，如图 3 所示。

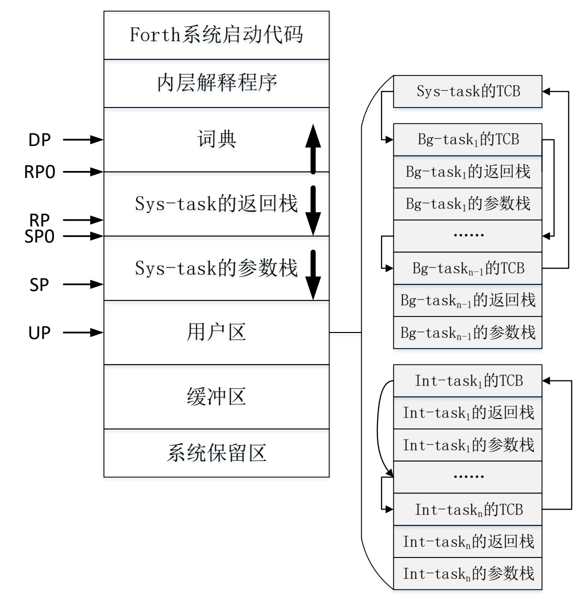


图 3 Forth 系统运行时的存储映像图

中断任务的 TCB 里定义了该任务运行时的信息，这些信息包括：任务的状态是就绪还是休眠；指向循环 TCB 链表中的下一个 TCB；堆栈从哪里开始，目前有多少条目；一个用于识别该任务是中断任务的标志。结构如表 1 所示。

表 1 Forth 系统中断任务的任务控制块结构

地址偏移	用途
0	status
2	follower
4	rp0
6	sp0
8	sp
10	TaskType

其中：status 描述的是任务的状态，存储的是任务就绪词 WAKE 或任务休眠词 PASS 的执行地址；follower 存储的是下一个中断任务的 TCB 的地址；rp0 描述的是当前中断任务的返回栈栈底指针；sp0 描述的是当前中断任务的参数栈栈底指针；sp 描述的是当前中断任务的参数栈栈顶指针；TaskType 的值为 1，描述的是该任务是中断任务，终端任务以及后台任务的 TCB 里也有这个标识，其值为 0。

(2)中断任务的创建

通过 INT-TASK:来创建一个中断任务。例如，用 INT-TASK:来创建一个中断任务 Int-task_1：32 32 INT-TASK: Int-task_1 .第一个 32 代表 Int-task_1 的返回栈空间有 32 个存储单元，第二个 32 代表 Int-task_1 的参数栈空间有 32 个存储单元。

采用 Forth2012 标准[14]，INT-TASK:的算法如下：

```
    : INT-TASK:
①   <builds
②   here ,
```

③ [s" /user" environment search-wordlist drop execute] literal

④ (rstacksize) allot here ,

⑤ (dstacksize) allot here ,

⑥ does> ;

①<builds 在词典里生成名称为 Int-task₁ 的名字域 NFA 词典条目，<builds 与 does>之间的词就是在创建中断任务 Int-task₁ 需要做的动作。名称 Int-task₁ 生成后，词典中下一个可用的存储单元就是 Int-task₁ 的参数域 PFA 的第一个字节的地址。以后执行 Int-task₁ 时，Int-task₁ 的参数域的第一个字节的地址会被保留在参数栈的栈顶。

②here 将用户区中下一个可用存储单元的地址放到参数栈栈顶，词 ，将该地址存到词典里 Int-task₁ 的参数域中。

③literal 在用户区里为 Int-task₁ 分配其 TCB 里地址偏移从 0~10 的地址空间。

④allot 在用户区里为中断任务 Int-task₁ 分配 32 个存储单元来作为返回栈空间，然后词 ，将用户区里下一个可用的存储单元的地址存在词典里 Int-task₁ 的参数域中。

⑤allot 在用户区里为中断任务 Int-task₁ 分配 32 个存储单元来作为参数栈空间，然后词 ，将用户区里下一个可用的存储单元的地址存在词典里 Int-task₁ 的参数域中。

Int-task₁ 的存储映像图如下图 4 所示：

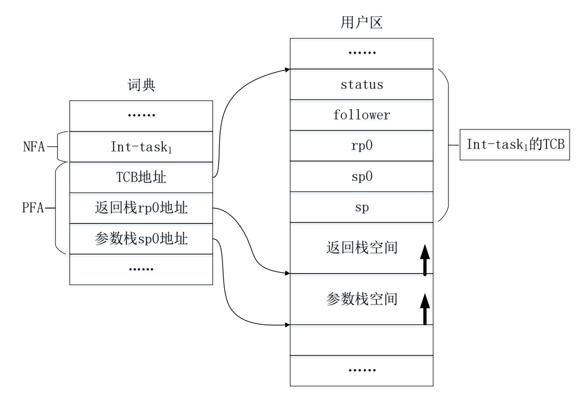


图 4　Int-task1 的存储映像图

(3)中断任务的运行

中断任务 Int-task₁ 建好后，还需要经过以下几个步骤才能得到运行：

a)初始化中断任务 Int-task₁ 的 TCB。将中断任务的参数栈指针和返回栈指针指向正确的位置，将中断任务的执行体（突发事件要做的事情）地址放到中断任务的返回栈栈顶，以便 Int-task₁ 运行的时候去执行这部分代码。

b)若 Int-task₁ 是新建的第一个中断任务，那么构建一个循环的中断任务队列，即让 Int-task₁ 的 TCB 中 follower 自己指向自己；若 Int-task₁ 不是建里的第一个中断任务，将中断任务 Int-task₁ 的 TCB 加入到 Forth 系统中原先存在的中断任务的任务队列里，构成一个循环的中断任务队列。

c)外部的突发事件通过中断请求的方式请求 CPU 处理，而中断任务是专门用于处理外部突发事件的，因此在中断服务程序里还要使中断任务 Int-task₁ 就绪，即将词 WAKE 的执行地址存入到 Int-task₁ 的 TCB 里的 status 存储单元中，以便任务调度的时候 Int-task₁ 能够被调度到。

d)执行任务调度原语 INT-PAUSE，使中断任务 Int-task₁ 得到 CPU 的控制权，处理突发事件。

2)任务调度原语 INT-PAUSE

由上文的论述可知，任务调度原语 INT-PAUSE 用于调度终端任务、后台任务及中断任务的运行，采用 Forth2012 标准，INT-PAUSE 算法如下：

① : **INT-PAUSE**

ENTER_CRITICAL

rp@ sp@ sp !

0 intTask @ > if branch1 else branch2 then

EXIT_CRITICAL ;

② : **branch1**

readyTask[intTask] @ tmpTCB !

1 intTask -!

tmpTCB @ dup @ i-cell+ >r ;

③ : **branch2**　0 readyTask[0] @ = if branch3 else branch4 then ;

④ : **branch3**　follower @ dup @ i-cell+ >r ;

⑤ : **branch4**

readyTask[0] @ tmpTCB !

0 readyTask[0] !

tmpTCB @ dup @ i-cell+ >r ;

:noname cell+ @ dup @ i-cell+ >r ; constant **PASS**

:noname up! sp@ sp! rp! ; constant **WAKE**

①INT-PAUSE 执行期间，不允许 CPU 响应中断，ENTER_CRITICAL 与 EXIT_CRITICAL 之间的代码是临界区。通过 rp@ sp@ sp ! 保存当前正在运行的任务的执行断点。若 intTask 大于 0，说明已经有中断任务就绪，然后去分支 branch1 中调度中断任务运行。若 intTask 等于 0，说明没有中断任务就绪，然后去分支 branch2 中寻找终端任务或者下一个就绪的后台任务运行。

②在 branch1 中，首先将 readyTask[intTask]中存储的就绪的中断任务的 TCB 地址赋值给 tmpTCB，然后 intTask 减 1，最后调度 tmpTCB 里存储的 TCB 运行，具体是当前正在运行的任务去执行存储在 tmpTCB 指向的 TCB 的 status 存储单元里的词 WAKE（就绪），在 WAKE 里将 CPU 的控制权交给就绪的中断任务。

③在 branch2 中，若 readyTask[0]等于 0，说明当前是在终端任务或者后台任务的执行体中发生任务调度，那么 branch3 的功能在于在终端-后台任务的任务链里寻找下一个已经就绪的任务，找到之后将 CPU 的控制权交给它。若 readyTask[0]不等于 0，说明当前是在中断任务的执行体中发生任务调度，那么 branch4 的功能在于返回发生中断时的终端任务或者后台任务，使终端-后台任务的任务链下一个就绪的任务运行。

④在 branch3 里，取出任务链里当前任务的下一个任务的 TCB 的 status 存储单元里的内容，若里面存的是词 PASS（休眠）的执行地址，则去执行 PASS，在 PASS 里继续往下寻找，直到在任务链里找到某个 status 是 WAKE 的任务，然后去执行 WAKE，在 WAKE 里将 CPU 的控制权交给找到的就绪的任务，之后恢复就绪任务的执行断点，最后就绪的任务开始执行。

⑤在 branch4 里，首先将 readyTask[0]中存储的就绪的终端任务或者后台任务的 TCB 地址赋值给 tmpTCB，然后 readyTask[0]赋值 0，最后调度 tmpTCB 里存储的 TCB 运行。

与基于 CPU 调度相比，对于基于虚拟机的 Forth 实时操作系统而言，其任务调度时现场保护仅需要将当前返回栈指针 rp 压入参数栈，并将当前参数栈指针 sp 保存到该任务的用户变量区里。而恢复现场仅需要从该任务的用户变量区里恢复 sp，并将参数栈栈顶值存入 rp 指针。

3　实验验证与分析

本文中 Forth 系统实时任务调度机制的具体实现基于开源的 AmForth。AmForth 是一个 16 位的 Forth 系统，其运行在 ATmega328 芯片上，ATmega328 是一个高性能、低功耗的 AVR 8 位微控制器，支持中断处理，时钟频率 16MHz，并且有 32KB 的 flash、2KB 的 RAM 以及 1KB 的 EEPROM 存储空间。ATmega328P 能够提供外部请求中断、引脚变化请求中断、定时器中断、串口中断等不同类型的中断。所有中断在中断向量表中都有一个单独的中断向量。这些中断具有与中断向量位置相同的优先级，中断向量地址越低，优先级越高。

由 2.1 节的分析可知，按照 INT-PAUSE 的调度方式，后台任务的数量并不会影响某个就绪的中断任务 Int-task_k 从就绪到运行需要等待的时间，中断任务 Int-task_k 从就绪到运行需要等待的时间与下列因素有关：a)终端任务或者后台任务从发生中断的地方执行到任务调度原语 INT-PAUSE 的时间；b)中断任务的数量以及中断任务一次执行完其执行体的时间；c)任务调度原语 INT-PAUSE 的执行时间。

在 AmForth 中新建一个后台任务 Bg-task_1，终端任务 Sys-task 的执行体由程序段②④⑤组成，后台任务 Bg-task_1 的执行体由程序段①③组成。突发事件在程序段①中通过中断的方式先后就绪了中断任务 Int-task_1~Int-task_5，中断任务 Int-task_1~Int-task_5 的执行体分别是程序段⑥~⑩，如图 5 所示。

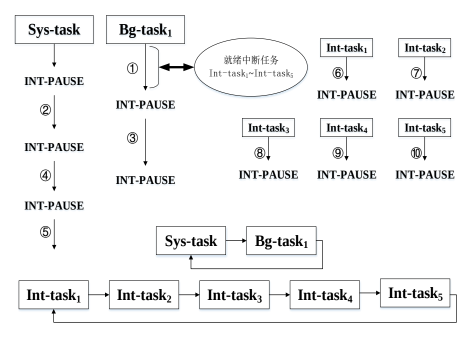


图 5　Sys-task、Bg-task 以及 Int-task 的任务链

后台任务 Bg-task_1 将程序段①执行完后，中断任务就绪表 readyTask[]的存储情况如图 6 所示。

	0	1	2	3	4	5
readyTask:	Bg-task1 的TCB地址	Int-task1 的TCB地址	Int-task2 的TCB地址	Int-task3 的TCB地址	Int-task4 的TCB地址	Int-task5 的TCB地址

intTask=5

图 6　readyTask[]的存储情况

表 2 给出了在 INT-PAUSE 调度方式下，中断任务 Int-task_1~Int-task_5 从就绪到运行需要等待的时间。

表 2　Int-task1~ Int-task5 从就绪到运行等待的时间　/ms

就绪的任务	从就绪到运行需要等待的时间
Int-task_1	18
Int-task_2	16
Int-task_3	12
Int-task_4	11
Int-task_5	9

$T_{INT\text{-}PAUSE}$ 表示任务调度原语 INT-PAUSE 的执行时间，T_1 表示后台任务 Bg-task_1 在程序段①中从发生中断的地方执行到 INT-PAUSE 所用的时间，由表 2 和 INT-PAUSE 的调度过程可以得出：

a)中断任务 Int-task_5 从就绪到运行需要等待的时间

$$T_{5-wait}=T_1+T_{INT-PAUSE}=9\text{ms}$$

b)$T_{5\text{-exec}}$ 表示中断任务 Int-task_5 执行其执行体所用的时间，中断任务 Int-task_4 从就绪到运行需要等待的时间

$$\begin{aligned}T_{4-wait}&=T_{5-wait}+T_{5-exec}+T_{INT-PAUSE}\\&=T_1+T_{5-exec}+2*T_{INT-PAUSE}\\&=11\text{ms}\end{aligned}$$

c)$T_{4\text{-exec}}$ 表示中断任务 Int-task_4 执行其执行体所用的时间，中断任务 Int-task_3 从就绪到运行需要等待的时间

$$\begin{aligned}T_{3-wait}&=T_{4-wait}+T_{4-exec}+T_{INT-PAUSE}\\&=T_1+\sum_{m=4}^{m=5}T_{m-exec}+3*T_{INT-PAUSE}\\&=12\text{ms}\end{aligned}$$

d)$T_{3\text{-exec}}$ 表示中断任务 Int-task_3 执行其执行体所用的时间，中断任务 Int-task_2 从就绪到运行需要等待的时间

$$\begin{aligned}T_{2-wait}&=T_{3-wait}+T_{3-exec}+T_{INT-PAUSE}\\&=T_1+\sum_{m=3}^{m=5}T_{m-exec}+4*T_{INT-PAUSE}\\&=16\text{ms}\end{aligned}$$

e)$T_{2\text{-exec}}$ 表示中断任务 Int-task_2 执行其执行体所用的时间，中断任务 Int-task_1 从就绪到运行需要等待的时间

$$\begin{aligned}T_{1-wait}&=T_{2-wait}+T_{2-exec}+T_{INT-PAUSE}\\&=T_1+\sum_{m=2}^{m=5}T_{m-exec}+5*T_{INT-PAUSE}\\&=18\text{ms}\end{aligned}$$

4 结束语

本文通过对 Forth 系统中多任务调度机制的相关关键问题进行研究，针对 Forth 系统没有实时调度去处理实时突发事件的问题，结合中断技术，本文新建了一种中断任务类型 INT-TASK:来处理突发事件。然后新定义了一个任务调度原语 INT-PAUSE 来实现 Forth 系统中终端任务、后台任务及中断任务的调度。从理论上提高了 Forth 系统对突发事件的实时响应性，并且在 AmForth 上通过实验验证了这一理论的正确性，推动了 Forth 系统多任务实时调度技术的发展，为基于虚拟机的嵌入式 Forth 系统大规模应用于对实时性有要求的嵌入式环境提供了一定的理论支撑和实现参考。

参考文献：

[1] Wikipedia. Charles H. Moore [EB/OL]. (2018) [2018-01-01]. http://en. wikipedia. org/wiki/Charles_H. _Moore.

[2] Leo B. Starting Forth [M]. Hermosa Beach: FORTH Inc, 1981: 1-5.

[3] FORTH Inc. SwiftForth IDE for Windows, Linux, and macOS [EB/OL]. (2018) [2018-1-12]. https://www. forth. com/embedded/.

[4] Hjrtland E, Chen L. EP32-A 32 bit Forth micorproprocessor [C]// Proc of CCECD. New York: Institute of Electrical and Electronics Engineers Inc, 2007: 518-521.

[5] Hanna D M, Jones B, Lorenz L, *et al.* An embedded Forth core with floating point and branch prediction [C]// Proc of the 56th International Midwest Symposium on Circuits and Systems. New York: Institute of Electrical and Electronics Engineers Inc, 2013: 1055-1058.

[6] Ting C H. VHDL Design of eP32 Microprocessor [EB/OL]. (2010) [2018-2-5]. http://www. forth. org/svfig/kk/08-2010-Ting. pdf.

[7] James B. The J1 Forth CPU [EB/OL]. (2018) [2018-1-10]. http://www. excamera. com/sphinx/fpga-j1. html.

[8] Don G. Forth Processor in VHDL [EB/OL]. (2018) [2018-1-15]. http://www. ultratechnology. com/4thvhdl. htm.

[9] John R. Using Forth as a VHDL [EB/OL]. (2018) [2018-1-15]. http://testra. com/Forth/VHDL. htm.

[10] John R. QSP16 [EB/OL]. (2018) [2018-1-12]. http://www. sandpipers. com/.

[11] Paweł G, Wojciech M Z. Tethered Forth system for FPGA applications [C]// Proc of Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2013.

[12] Chuck M. GreenArrays [EB/OL]. (2018) [2018-1-15]. http://www. greenarraychips. com/.

[13] 代红兵. 高效微机实时多任务操作系统设计与实现 [J]. 中国科学院研究生院学报, 1993, 10 (3): 283-292. (Dai Hongbing. The design and realization of efficent microcomputer operating system of real-time multitask [J]. Journal of the Graduate School of the Chines Academy of Sciences, 1993, 10 (3): 283-292.)

[14] Forth200x committee. Forth 2012 Standard [EB/OL]. (2018) [2018-3-12]. http://forth-standard. org/.